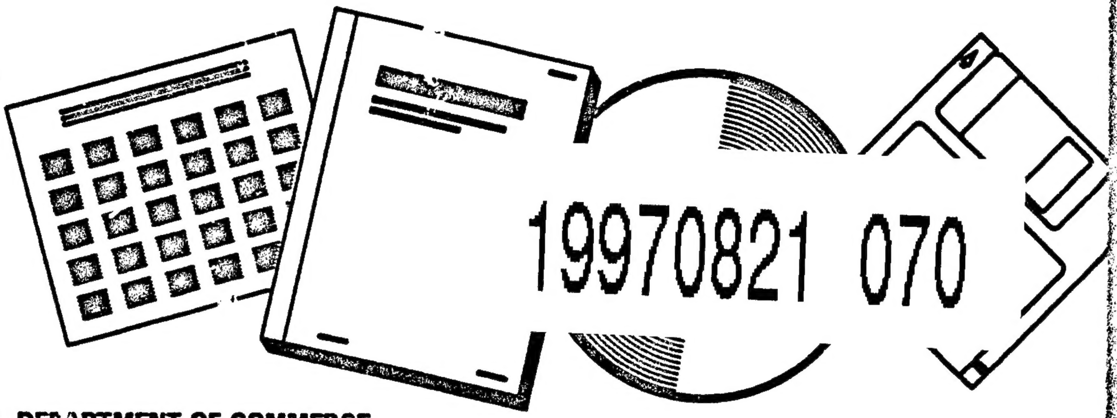
COMPLETING THE TEMPORAL PICTURE

STANFORD UNIV., CA

DEC 89



U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

BIBLIOGRAPHIC INFORMATION

PB96-149737

Report Nos: STAN-CS-89-1296

Title: Completing the Temporal Picture.

Date: cDec 89

Authors: Z. Manna and A. Pnueli.

Performing Organization: Stanford Univ., CA. Dept. of Computer Science.**Weizmann Inst. of Science, Rehovoth (Israel). Dept. of Applied Mathematics.

Sponsoring Organization: *National Science Foundation, Washington, DC.*Defense Advanced Research Projects Agency, Arlington, VA.*Air Force Office of Scientific Research, Bolling AFB, DC.

Contract Nos: NSF-DCR-8413230, NSF-CCR-8812595, DARPA-N00039-84-C-0211, AFOSR-87-01 AFOSR-88-0281

NTIS Field/Group Codes: 62B (Computer Software)

Price: PC A03/MF A01

Availability: Available from the National Technical Information Service, Springfiel VA. 22161

Number of Pages: 30p

Keywords: *Computer program verification, *Mathematical logic, Numerical analysis, Algorithms, Computations, *Temporal logic.

Abstract: The paper presents a relatively complete proof system for proving the validity of temporal properties of reactive programs. The presented proof system improves on previous temporal systems, such as (MP83a) and (MP83b), in that it redu the validity of program properties into pure assertional reasoning, not involving additional temporal reasoning. The proof system is based on the classification of temporal properties according to the Borel hierarchy, providing an approriate proof rule for each of the main classes, such as safety, response, and progress propertie

December 1989

Report No. STAN-CS-89-1296

Completing the Temporal Picture

by

Zohar Manna and Amir Pnueli

Department of Computer Science

Stanford University
Stanford, California 94305

REPRODUCED BY: NTIS
U.S. Department of Commerce
National Technical Information Service
Springfield, Virginia 22161

Completing the Temporal Picture*

Zohar Manna
Stanford University [†]
and
Weizmann Institute of Science[‡]

Amir Pnueli
Weizmann Institute of Science[‡]

Abstract

The paper presents a relatively complete proof system for proving the validity of temporal properties of reactive programs. The presented proof system improves on previous temporal systems, such as [MP83a] and [MP83b], in that it reduces the validity of program properties into pure assertional reasoning, not involving additional temporal reasoning. The proof system is based on the classification of temporal properties according to the Borel hierarchy, providing an appropriate proof rule for each of the main classes, such as *safety*, *response*, and *progress* properties.

1 Introduction

Temporal logic is, by now, one of the acceptable and frequently used approaches to the formal specification and verification of concurrent and reactive programs. Even though we have witnessed, over the last several years, a great progress in the automatic verification of finite-state programs, the main tool for establishing that a proposed implementation satisfies its temporal specification is still that of deductive verification, using a set of axioms and inference rules.

As described in [MP83a] (see also [MP83b] and [Pnu86]), a proof system that supports the verification of temporal properties over reactive programs has to deal with three types of validity.

- $\mathcal{A}$– *Assertional* Validity. This is the validity of non-temporal (state) formulae (also called *assertions*) over an arbitrary interpretation.
- $\mathcal{T}$– *General Temporal* Validity. This is the validity of temporal formulae over arbitrary sequences of states (models).
- $\mathcal{P}$– *Program* Validity. This is the validity of temporal formulae over sequences of states which represent computations of the analyzed program.

*This research was supported in part by the National Science Foundation under grants DCR-8413230 and CCR-8812595, by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, and by the United States Air Force Office of Scientific Research under contracts AFOSR 87-0149 and 88-0281.

[†] Department of Computer Science, Stanford University, Stanford, CA 94305

[‡] Department of Applied Mathematics, Weizmann Institute, Rehovot, Israel

Corresponding to these three types of validity, the proof system may be partitioned into three parts, each providing axioms and rules for establishing the validity of the corresponding type. This is essentially the structure of the proof system presented in [MP83b], where we refer to the assertional part as the *domain* part.

The program part presents some basic proof rules and some derived rules. The derived rules provide direct support for proving some of the most frequently used temporal properties of programs.

One group of rules establishes the validity of the *invariance* formulae $\Box q$ and $\Box(p \rightarrow \Box q)$, which express the invariance of a state formula q, either throughout the computation, or triggered by the occurrence of p.

Another group of rules establishes the validity of the *eventuality* formulae $\Diamond q$ and $\Box(p \rightarrow \Diamond q)$, which express the guarantee that q will eventually happen, either once or following each occurrence of p.

These proof rules are completely satisfactory for establishing this restricted but very prevalent set of temporal formulae. The rules derive temporal conclusions from assertional premises. They have been proven relatively complete, and are the main working tools for verification of the temporal properties of programs (see, e.g., [OL82], [MP84], [Krö87]).

However, the question which is only partially answered in [MP83a] is how do we prove all the other properties whose expression in temporal logic does not fall into the restricted class of invariance and eventuality formulae. The partial solution given there is a general relative completeness theorem, which shows that the program part is adequate for reducing the validity of a temporal formula over a given program ($\mathcal{P}$-validity) into a set of valid formulae, which are either assertional ($\mathcal{A}$-valid), or temporal but valid over arbitrary sequences of states ($\mathcal{T}$-valid).

We remind the reader that this is the general character of all *relative* completeness results for program logics such as Hoare logic ([Apt81]) or Dynamic Logic ([Har79]). Since, as soon as we consider programs that operate over infinite domains, we lose the possibility of having true completeness, the best we can hope for is relative completeness ([Coo78]). This type of completeness ensures an effective reduction from the validity of a program logic statement into the validity of a finite number of assertional statements.

Unfortunately, the reduction given in [MP83a] is not only into assertional statements, but also into generally ($\mathcal{T}$-) valid temporal statements. This requires a proof of a general program property to be based not only on assertional reasoning, but also on *temporal reasoning*, which is less familiar, even to a person who is well versed in general logic. This fact has been considered by some researchers a deficiency, and has caused them to shy away from the temporal proof system and look for alternative formalisms, in which a complete reduction into assertional statements is guaranteed ([AS89], and also see [MP87]).

In this paper we attempt to remedy the situation by providing a richer proof system for the program part, which ensures complete reduction of a general temporal formula (given in a canonical form) into a finite set of assertional statements, whose validity imply the validity of the original temporal formula.

The approach to a complete proof system is based on a classification of temporal properties according to their expression in a *canonical form*, which applies a set of restricted future modalities to arbitrary past formulae. This classification establishes a hierarchy of temporal properties ([MP89]), whose classes can be characterized according to three different criteria. We have already mentioned their characterization in terms of the syntactic form of their canonical representation.

Another characterization is semantical, looking at a property as the set of all sequences which have this property. By this view we can give a topological characterization to the classes in our hierarchy, locating it at the first two levels of the Borel hierarchy. The third characterization is in terms of structural restrictions on the Streett automaton that recognizes precisely the set of the infinite sequences which have the property.

In principle, we should provide a separate proof rule for each of the property classes in our hierarchy. In practice, we concentrate on three particular classes, which have special significance as expressing most of the interesting program properties, and forming a natural generalization of the two classes of invariance and eventuality properties considered in the previous proof systems. These are the classes of:

- *Safety* Properties. These are all the properties that can be expressed by a temporal formula of the form
$$\Box q$$
for some *past* formula q.
- *Response* Properties. These are all the properties that can be expressed by a temporal formula of the form
$$\Box(p \rightarrow \Diamond q), \text{ or alternately, } \Box\Diamond q$$
for some *past* formulae p and q.
- *Progress* Properties. These are all the properties that can be expressed by a temporal formula of the form
$$\Box\Diamond p \rightarrow \Box\Diamond q$$
for some *past* formulae p and q.

We provide complete rules for each of these classes. This provides full coverage for the entire temporal logic, since by [LPZ85] (see also [Tho81]), any temporal formula φ is equivalent to a conjunction of progress properties. Therefore, to prove the $\mathcal{P}$-validity of φ, it is sufficient to prove the $\mathcal{P}$-validity of each of the conjuncts, for which we can use the rule for progress properties.

2 Programs and Computations

The basic computational model we use to represent programs is that of a *fair transition system*. In this model, a program P consists of the following components.

- $V = \{u_1, ..., u_n\}$ - A finite set of *state variables*. Some of these variables represent *data* variables, which are explicitly manipulated by the program text. Other variables are *control* variables, which represent, for example, the location of control in each of the processes in a concurrent program. We assume each variable to be associated with a domain, over which it ranges.
- Σ - A set of *states*. Each state $s \in \Sigma$ is an interpretation of V, assigning to each variable $y \in V$ a value over its domain, which we denote by $s[y]$.

- $\mathcal{T}$ - A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is associated with an assertion $\rho_\tau(V, V')$, called the *transition relation*, which refers to both an unprimed and a primed version of the state variables. The purpose of the transition relation ρ_τ is to express a relation between a state s and its successor s'. We use the unprimed version to refer to values in s, and the primed version to refer to values in s'. For example, the assertion $x' = x + 1$ states that the value of x in s' is greater by 1 than its value in s.

- Θ - The *precondition*. This is an assertion characterizing all the initial states, i.e., states at which the computation of the program can start. A state is defined to be *initial* if it satisfies Θ.

- $\mathcal{C} = \{C_1, ..., C_r\}$ - A finite set of *continual fairness* requirements (also called *justice* or *weak fairness* requirements). Each continual fairness requirement $C_i \in \mathcal{C}$ consists of two sets of transitions $C_i = (E_i, T_i)$, $E_i \subseteq T_i \subseteq \mathcal{T}$, on which the requirement of continual fairness is imposed. Intuitively, the continual fairness requirement $(E_i, T_i) \in \mathcal{C}$ disallows a computation in which, beyond a certain point, E_i is continually enabled, but no transition of T_i is taken beyond this point.

- $\mathcal{R} = \{R_1, ..., R_t\}$ - A family of *recurrent fairness* requirements (also called *strong fairness* requirements). Each recurrent fairness requirement $R_i \in \mathcal{R}$ consists of two sets of transitions $R_i = (E_i, T_i)$, $E_i \subseteq T_i \subseteq \mathcal{T}$, on which the requirement of recurrent fairness is imposed. Intuitively, the recurrent fairness requirement $(E_i, T_i) \in \mathcal{R}$ disallows a computation in which, beyond a certain point, E_i is enabled infinitely many times, but no transition of T_i is taken beyond that point.

We define the state s' to be a *τ-successor* of the state s if

$$\langle s, s' \rangle \models \rho_\tau(V, V'),$$

where $\langle s, s' \rangle$ is the joint interpretation which interprets $x \in V$ as $s[x]$, and interprets x' as $s'[x]$. Following this definition, we can view the transition τ as a function $\tau : \Sigma \mapsto 2^\Sigma$, defined by:

$$\tau(s) = \{s' \mid s' \text{ is a } \tau\text{-successor of } s\}.$$

We say that the transition τ is *enabled* on the state s, if $\tau(s) \neq \phi$. Otherwise, we say that τ is *disabled* on s. We say that a state s is *terminal* if all the transitions $\tau \in \mathcal{T}$ are disabled on it. The enabledness of a transition τ can be expressed by the formula

$$En(\tau) : \ (\exists V')\rho_\tau(V, V'),$$

which is true in s iff s has some τ-successor.

For a set of transitions $E \subseteq \mathcal{T}$, we say that E is *enabled* on s, denoted by $En(E)$, if *some* transition $\tau \in E$ is enabled on s, and that E is *disabled* on s if *all* transitions $\tau \in E$ are disabled on s.

Given a program P for which the above components have been specified, we define a *computation* of P to be a finite or infinite sequence of states $\sigma : s_0, s_1, s_2, ...,$ satisfying the following requirements:

- *Initiality* $\quad$ s_0 is initial, i.e., $s_0 \models \Theta$.

- *Consecution* For each $j = 0, 1, \ldots$ the state s_{j+1} is a τ-successor of the state s_j, i.e., $s_{j+1} \in \tau(s_j)$, for some $\tau \in \mathcal{T}$. In this case, we say that the transition τ is *taken* at position j in σ. For a set of transitions $T \subseteq \mathcal{T}$, we say that T is *taken* at position j, if some $\tau \in T$ is taken at j.
- *Termination* Either σ is infinite, or it ends in a state s_k which is terminal.
- *Continual Fairness* For each $(E_i, T_i) \in \mathcal{C}$ it is required that, if E_i is continually enabled beyond some point in σ, then T_i must be taken at infinitely many positions in σ.
- *Recurrent Fairness* For each $(E_i, T_i) \in \mathcal{R}$ it is required that, if E_i is enabled on infinitely many states of σ, then T_i must be taken at infinitely many positions in σ.

For a program P, we denote by $Comp(P)$ the set of all computations of P. For simplicity, we will only consider programs for which $\mathcal{T}$ is always enabled. Such programs have only infinite computations.

3 Temporal Logic

We assume an underlying assertional language, which contains the predicate calculus, and interpreted symbols for expressing the standard operations and relations over some concrete domains. For the sake of completeness, we require that one of the domains is that of the integers, or another domain with similar expressive power. We refer to a formula in the assertional language as a *state formula*, or simply as an *assertion*.

A *temporal formula* is constructed out of state formulae to which we apply the boolean operators $\neg$ and $\vee$ (the other boolean operators can be defined from these), and the following basic temporal operators:

$\bigcirc$ - Next $\odot$ - Previous
$\mathcal{U}$ - Until $\mathcal{S}$ - Since

A *model* for a temporal formula p is a finite or infinite sequence of states $\sigma : s_0, s_1, \ldots$, where each state s_j provides an interpretation for the variables mentioned in p. For simplicity, we will only consider the case of infinite models.

Given a model σ, as above, we present an inductive definition for the notion of a temporal formula p holding at a position $j \geq 0$ in σ, denoted by $(\sigma, j) \models p$.

- For a state formula p,

$$(\sigma, j) \models p \iff s_j \models p.$$

That is, we evaluate p locally, using the interpretation given by s_j.

- $(\sigma, j) \models \neg p \iff (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \iff (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \bigcirc p \iff (\sigma, j+1) \models p$
- $(\sigma, j) \models p \mathcal{U} q \iff$ for some $k \geq j, (\sigma, k) \models q$, and for every i such that $j \leq i < k, (\sigma, i) \models p$
- $(\sigma, j) \models \odot p \iff j > 0$ and $(\sigma, j-1) \models p$
- $(\sigma, j) \models p \mathcal{S} q \iff$ for some $k \leq j, (\sigma, k) \models q$, and for every i such that $j \geq i > k, (\sigma, i) \models p$

Additional temporal operators can be defined as follows:

$\Diamond p = \mathrm{T}\,\mathcal{U} p$	- Eventually	$\diamondminus p = \mathrm{T}\,\mathcal{S} p$	- Sometimes in the past
$\Box p = \neg\Diamond\neg p$	- Henceforth	$\boxminus p = \neg\diamondminus\neg p$	- Always in the past
$p \mathsf{U} q = \Box p \vee (p \mathcal{U} q)$	- Unless	$p \mathsf{S} q = \boxminus p \vee (p \mathcal{S} q)$	- Weak Since

Another useful derived operator is the *entailment* operator, defined by:

$$p \Rightarrow q \quad \Longleftrightarrow \quad \Box(p \rightarrow q).$$

A formula that contains no future operators is called a *past* formula. A formula that contains no past operators is called a *future* formula. Note that a state formula is both a past and a future formula. We refer to a past formula [future formula] that is not also a state formula, as a *strict-past* [*strict-future*, respectively] formula. For a state formula p and a state s such that p holds on s, we say that s is a *p-state*.

If $(\sigma, 0) \models p$, we say that p *holds* on σ, and denote it by $\sigma \models p$. A formula p is called *satisfiable* if it holds on some model. A formula is called (temporally) *valid* if it holds on all models.

Two formulae p and q are defined to be *equivalent* if the formula $p \equiv q$ is valid, i.e., $\sigma \models p$ iff $\sigma \models q$, for all σ.

The notion of validity defined above is the notion of $\mathcal{T}$-validity. Given a program P, we can restrict our attention to the set of models which correspond to computations of P, i.e., *Comp*(P). This leads to the notion of $\mathcal{P}$-validity, by which p is *P-valid* if it holds over all the computations of P. Similarly, we obtain the notions of P-satisfiability and P-equivalence.

Canonical Form and Classification

By [LPZ85] (see also [Tho81]), every temporal formula is equivalent to a formula of the form

$$\bigwedge_{i=1}^{n} (\Box\Diamond p_i \vee \Diamond\Box q_i),$$

for some past formulae $p_i, q_i, i = 1, ..., n$.

Based on this canonical form we can classify the properties expressible by temporal logic according to their expressibility by restricted cases of this general formula. We list below the main classes in this classification, specifying their temporal characterizations. For each class we present the form of the temporal formulae that express the properties in that class, where the subformulae p, q, p_i, q_i appearing there are arbitrary past formulae. We refer the reader to [MP89] for additional properties and characterizations of this hierarchy.

- *Safety Properties* – $\Box p$.
- *Termination Properties* – $\Diamond p$.
- *Intermittence Properties* – $\Box p \vee \Diamond q$.
- *Multiple Intermittence Properties* – $\bigwedge_{i=1}^{n}(\Box p_i \vee \Diamond q_i)$.
- *Response Properties* – $\Box\Diamond p$.

- *Persistence Properties* - $\Diamond\square p$.
- *Progress Properties* - $\square\Diamond p \vee \Diamond\square q$.
- *Multiple Progress Properties* - $\bigwedge_{i=1}^{n}(\square\Diamond p_i \vee \Diamond\square q_i)$.

As stated above, the multiple progress class is the maximal class of properties expressible by temporal logic.

4 Rules for Safety

From now on, we fix our attention on a program P, specified by the components $\langle V, \Sigma, \mathcal{T}, \Theta, \mathcal{C}, \mathcal{R}\rangle$.

In this section we consider proof rules for establishing the $\mathcal{P}$-validity of a safety formula. As we recall, a safety formula has the form $\square p$ for some past formula p. Let us review first the appropriate rule for the simpler case that p is a state formula.

For a transition τ, and state formulae p and q, we define the *verification condition* of τ, relative to p and q, to be the implication:

$$(\rho_\tau \wedge p) \rightarrow q', \text{ denoted by } \{p\}\tau\{q\},$$

where ρ_τ is the transition assertion corresponding to τ, and q', the *primed version* of the assertion q, is obtained from q by replacing each variable occurring in q by its primed version. Since ρ_τ holds for two states s and s' iff s' is a τ-successor of s, and q' states that q holds on s', it is not difficult to see that

> If the verification condition $\{p\}\tau\{q\}$ is assertionally valid, then every τ-successor of a p-state is a q-state.

For a set of transitions $T \subseteq \mathcal{T}$, we denote by $\{p\}T\{q\}$ the verification condition of T, relative to p and q, requiring that $\{p\}\tau\{q\}$ holds for *every* $\tau \in T$.

The following rule is sound and complete for establishing the P-validity of the invariance formula $\square q$ for a state formula q, over the program P.

$$\boxed{\begin{array}{lll} \textbf{INV} & \text{I1.} & \Theta \rightarrow \varphi \\ & \text{I2.} & \varphi \rightarrow q \\ & \text{I3.} & \{\varphi\}\mathcal{T}\{\varphi\} \\ \hline & & \square q \end{array}}$$

This rule uses an auxiliary assertion φ which, by premise I1, holds initially, and by premise I3 is propagated from each state to its successor. This shows that φ is an invariant of the program, that is, it holds continuously over all computations of P. Since, by I2, the assertion φ implies q, it follows that q is also an invariant of the program.

Generalizing to Past Formulae

Next, we have to extend the INV rule to deal with formulae q, which are past formulae. First, we extend the notion of the primed version of a formula, to apply also to a past formula. Recall that the intended meaning of a primed formula is to express the value of a formula in the next state, in terms of the values of the variables in the next state and in terms of values in the current state. This is inductively defined as follows:

- For a state formula $p(V)$, we define as before

$$(p(V))' = p(V').$$

- For a *previous* formula

$$(\odot p)' = p.$$

This corresponds to our intuition that $\odot p$ holds in the next state iff p holds now.

- For a *since* formula

$$(p\mathcal{S}q)' \;=\; q' \vee ((p\mathcal{S}q) \wedge p').$$

This corresponds to the intuition that $p\mathcal{S}q$ holds in the next state if, either q holds there, or $p\mathcal{S}q$ holds now and p holds next.

With this definition, we extend the notion of the verification condition $\{p\}\tau\{q\}$ to apply also to past formulae p and q, and to mean

$$(\rho_\tau \wedge p) \Rightarrow q'.$$

Note that since we work with temporal formulae, we replaced the previous implication by an entailment, because we expect the implication to hold at *all* positions of the computation, not only at the first one.

With this extension, the general single rule for establishing safety properties is given by

$$\boxed{\begin{array}{lll} \textbf{SAFE} & \text{S1.} & (\Theta \wedge \textbf{first}) \Rightarrow \varphi \\ & \text{S2.} & \varphi \Rightarrow q \\ & \text{S3.} & \{\varphi\}\mathcal{T}\{\varphi\} \\ \hline & & \Box q \end{array}}$$

The implications, appearing in the premises I1 and I2 of the INV rule, have been replaced in the SAFE rule by the entailments, appearing in the premises S1 and S2. In S1 we also added the conjunct first which is an abbreviation for the formula $\neg\odot\mathrm{T}$, characterizing the first position in the computation as the only position that has no predecessor. This conjunct is sometimes necessary to ensure that φ holds in the first position.

Examining the premises S1 – S3 of the SAFE rule, we observe that they all have the form of temporal formulae, which are actually other safety formulae. How are these to be proven? It seems that we need some additional rules, belonging to the general part. These rules enable us to prove some temporal formulae that are generally valid, i.e., hold over any sequence of states, unrelated to any particular program.

The first rule we consider is the rule of temporal instantiation, which provides a basic tool for deriving temporal validities from assertional ones. Let q be a state formula containing the propositional symbol p, and let φ be a temporal formula. We denote by $q[\varphi/p]$ the temporal formula obtained from q by replacing all occurrences of p by φ.

$$\boxed{\textbf{INST} \quad \frac{q}{\Box q[\varphi/p]}}$$

Note, in particular, that if q has the form $t \rightarrow r$ then the temporal conclusion is an entailment of the form $t[\varphi/p] \Rightarrow r[\varphi/p]$. This rule is often used, without any instantiation, to derive the temporal validity of $\Box q$ from the assertional validity of q. In these cases, it is sometimes referred to as *generalization.*

The next rule we consider can be viewed as stating the monotonicity of the temporal operator $\Box$. For two temporal formulae p and q, we can interpret the entailment $p \Rightarrow q$, i.e., $\Box(p \rightarrow q)$, as an ordering relation between the formulae, stating that p is smaller (stronger) than q. Indeed, for a sequence σ, $p \Rightarrow q$ claims that the set of positions at which p holds is contained in the set of positions at which q holds. Monotonicity of the $\Box$ operator states that if $p \Rightarrow q$, and $\Box p$ is valid, then so is $\Box q$.

$$\boxed{\textbf{S-MON} \quad \begin{array}{ll} \text{A1.} & p \Rightarrow q \\ \text{A2.} & \Box p \\ \hline & \Box q \end{array}}$$

This rule can also be viewed as a temporal version of Modus Ponens, where entailment replaces implication. In fact, the two preceding rules provide a formal support for many elementary manipulations, such as substituting equals for equals, and using any instantiation of propositional tautologies. We refer to any such manipulation as justified by *propositional reasoning.*

In addition to these very general rules, we need in our general part some properties which are specific to the initial part of a sequence of states. These will enable us to draw some conclusions from the formula **first**, as is needed in premise S1 of the SAFE rule.

These are presented by the following two axioms:

- I-PREV: $\textbf{first} \Rightarrow \neg \ominus p$
- I-SINCE: $\textbf{first} \Rightarrow \big((p \mathcal{S} q) \equiv q\big)$

The I-PREV axiom states that no *previous* formula can hold at the initial position of any sequence. The I-SINCE axiom states that the formula $p\mathcal{S}q$ can hold at the initial position iff q holds there.

The Completeness of the SAFE Rule

We proceed to consider the applicability of the **SAFE** rule to the proofs of safety properties. First, we present an example, illustrating its use.

Example 4.1 Consider the trivial program with a single state variable x, precondition $x = 0$, and a single transition τ whose assertion is given by $\rho_\tau : x' = x + 1$. Observe that this program has a single infinite computation, given by $\langle x : 0\rangle, \langle x : 1\rangle, \langle x : 2\rangle, \ldots$

We wish to prove for this program the trivial safety property

$$\Box((x = 10) \rightarrow \diamondminus(x = 5)).$$

This property claims that any state in which $x = 10$ must have been preceded by a state in which $x = 5$. Note that this trivial property would not be true for a program that advances in steps of 2, rather than steps of 1.

To prove this property, we identify q as $(x = 10) \rightarrow \diamondminus(x = 5)$ and intend to use the **SAFE** rule. As the auxiliary formula φ, we take $(x \geq 5) \rightarrow \diamondminus(x = 5)$. The rule requires showing the following three premises:

S1. $[(x = 0) \;\wedge\; \mathbf{first}] \Rightarrow [(x \geq 5) \rightarrow \diamondminus(x = 5)]$

S2. $((x \geq 5) \rightarrow \diamondminus(x = 5)) \Rightarrow ((x = 10) \rightarrow \diamondminus(x = 5))$

S3. $[(x' = x + 1) \;\wedge\; ((x \geq 5) \rightarrow \diamondminus(x = 5))] \Rightarrow [(x' \geq 5) \rightarrow (\diamondminus(x = 5) \vee (x' = 5))]$

In S3 we have already expanded $(\diamondminus(x = 5))'$ into $(\diamondminus(x = 5) \vee (x' = 5))$. All of these apparently temporal formulae can be established by the **INST** rule, using the following three valid state formulae, and their associated instantiations.

V1. $((x = 0) \wedge p) \rightarrow ((x \geq 5) \rightarrow r)$

with the replacement of $(\mathbf{first}, \diamondminus(x = 5))$ for the proposition symbols (p, r), respectively.

V2. $((x \geq 5) \rightarrow p) \rightarrow ((x = 10) \rightarrow p)$

with the replacement of $\diamondminus(x = 5)$ for the proposition symbol p.

V3. $[(x' = x + 1) \;\wedge\; ((x \geq 5) \rightarrow p)] \rightarrow [(x' \geq 5) \rightarrow (p \vee (x' = 5))]$

with the replacement of $\diamondminus(x = 5)$ for the proposition symbol p.

Theorem 7.2, presented in Section 7, establishes the adequacy of the **SAFE** rule by stating:

*The **SAFE** rule is complete, relative to assertional validity, for proving the $\mathcal{P}$-validity of any safety property.*

The proof of the theorem is based on the construction of a big past invariant which relates the values of variables in an accessible state (i.e., appearing in some computation of P) to the boolean values of the temporal sub-formulae of the past formula q, whose invariance we wish to establish.

Causality Formulae

Even though, in theory, the completeness theorem above fully settles the question of proving the validity of safety formulae, there is a practical interest in identifying special forms of safety formulae, for which a specific proof methodology exists. One of these subclasses contains the properties expressible by the *causality* formula

$$p \Rightarrow \diamondminus q$$

for past formulae p and q. The causality formula states that every p-state is necessarily preceded by a q-state.

To present a proof rule for causality properties, we define first the *inverse verification condition*, denoted by $\{p\}\tau^{-1}\{q\}$ and standing for the entailment

$$(\rho_\tau \wedge p') \Rightarrow q.$$

The validity of this condition ensures that any τ-predecessor of a p-state must be a q-state. The condition is extended to sets of transitions $T \subseteq \mathcal{T}$ in the usual way. Then, the following rule is adequate for proving causality properties.

$$\boxed{\begin{array}{lll} \textsc{caus} & \text{K1.} & p \Rightarrow (\varphi \vee q) \\ & \text{K2.} & (\Theta \wedge \textbf{first}) \Rightarrow \neg\varphi \\ & \text{K3.} & \{\varphi\}\mathcal{T}^{-1}\{\varphi \vee q\} \\ \hline & & p \Rightarrow \diamondminus q \end{array}}$$

By premise K1, any state satisfying p, either already satisfies q, or satisfies the auxiliary past formula φ. By premise K3, the predecessor of any φ-state must satisfy $\varphi \vee q$. Thus, if we do not find a q preceding p, φ propagates all the way to the initial position. However, this contradicts premise K2, according to which the initial position cannot satisfy φ.

Incremental Proofs

In the previous paragraphs, we have considered how to establish the invariance of some past formulae. Having established some basic invariants of this form, we may want to use them in order to derive more complex properties. For this purpose, we quote again the **S-MON** rule, which suggests a strategy, to which we refer as the *incrementality principle*. According to this principle, we establish first the validity of a simpler safety property $\Box p$. Later, whenever we have to establish the validity (over P) of a premise that has the form $\Box\psi$, we can instead establish the validity of $p \Rightarrow \psi$.

5 Rules for Response

Response properties are those which can be expressed by a formula of the form

$$p \Rightarrow \Diamond q, \text{ or equivalently } \Box(p \rightarrow \Diamond q)$$

for some past formulae p and q. Now that we have learned, in the previous section, how to generalize rules having assertional premises into rules with temporal premises involving past formulae, it is straightforward to properly adapt the set of rules from [MP83a]. The rules for establishing response properties can be partitioned into *single-step* rules and *extended* rules. We consider each group in turn.

Rules for Single-Step Response

These are the rules that establish properties that depend on the execution of a single helpful transition (which may be selected out of several candidates) to accomplish the guaranteed response q. We have three rules in this group, which differ by the type of *fairness* on which they rely.

The first rule is unconditional of any fairness assumption, and only relies on the fact that as long as there are enabled transitions, some transition will eventually be taken.

$$\begin{array}{lll} \textbf{B-RESP} & \text{B1.} & p \Rightarrow (q \lor \varphi) \\ & \text{B2.} & \{\varphi\} T \{q\} \\ & \text{B3.} & \varphi \Rightarrow (q \lor En(T)) \\ \hline & & p \Rightarrow \Diamond q \end{array}$$

The rule considers three past formulae p, q, and the auxiliary φ. Premise B1 requires that any p-state, either already satisfies q, or satisfies φ. Premise B2 requires that taking any transition from a φ-state, must lead to a q-state. Premise B3 requires that at least one transition must be enabled on each φ-state that does not satisfy q. Clearly such a transition must be taken next, resulting in a q-state.

The next single-step rule relies on continual fairness to ensure that eventually a helpful transition, leading to q, will be taken. It assumes a continual fairness requirement $(E, T) \in C$.

$$\begin{array}{lll} \textbf{C-RESP} & \text{C1.} & p \Rightarrow (q \lor \varphi) \\ & \text{C2.} & \{\varphi\} T \{q \lor \varphi\} \\ & \text{C3.} & \{\varphi\} T \{q\} \\ & \text{C4.} & \varphi \Rightarrow (q \lor En(E)) \\ \hline & & p \Rightarrow \Diamond q \end{array}$$

Premise C1 ensures, as before, that p entails q or φ. Premise C2 states that any transition of the program, either leads from φ to q, or preserves φ. Premise C3 states that any transition in the helpful set T leads from φ to q. Premise C4 ensures that E is enabled as long as φ holds and q does not occur. It is not difficult to see that if p happens, but is not followed by a q, then φ must hold continuously beyond this point, and no transition of T is taken. However, due to C4, this means that E is continuously enabled beyond this point, which violates the requirement of continual fairness represented by (E, T).

The last rule relies on a recurrent fairness requirement $(E, T) \in \mathcal{R}$.

$$\begin{array}{lll} \textbf{R-RESP} & \text{R1.} & p \Rightarrow (q \lor \varphi) \\ & \text{R2.} & \{\varphi\} T \{q \lor \varphi\} \\ & \text{R3.} & \{\varphi\} T \{q\} \\ & \text{R4.} & \varphi \Rightarrow \Diamond (q \lor En(E)) \\ \hline & & p \Rightarrow \Diamond q \end{array}$$

The difference between this rule and its c-version is in the fourth premise. While C4 requires that φ entails the occurrence of q or the enabling of E *now*, R4 requires the *eventual* occurrence of q or enabling of E. Here, an occurrence of p not followed by a q, leads, as before, to φ holding continuously, and no transition of T being taken. However, the weaker premise R4 guarantees that E is enabled infinitely many times, which suffices to violate the recurrent fairness requirement (E,T).

In view of the fact that premise R4 appears to be of the same form as the conclusion, i.e., another response formula, one may wonder whether we may not enter a circular loop, trying to prove one response property by another. The answer to this problem is that when we prove premise R4, we actually consider a simpler program, in which none of the transitions of E is ever used. This is because the first time a transition of E can be taken, we have already achieved the goal of a state on which E is enabled.

Rules for Extended Response

These rules combine single-step response properties to form general response properties, which need more than a single helpful transition for their achievement.

First, we list two basic rules, which express the monotonicity and transitivity of response properties. They properly belong to the general part.

$$\boxed{\textbf{R-MON}\quad \begin{array}{c} p \Rightarrow q\ ,\ r \Rightarrow t \\ q \Rightarrow \Diamond r \\ \hline p \Rightarrow \Diamond t \end{array}} \qquad \boxed{\textbf{R-TRNS}\quad \begin{array}{c} p \Rightarrow \Diamond q \\ q \Rightarrow \Diamond r \\ \hline p \Rightarrow \Diamond r \end{array}}$$

The most important rule for establishing extended response properties is based on well-founded induction.

We say that the binary relation $\succ$ over the set $\mathcal{A}$ (often presented as the pair $(\mathcal{A},\succ)$) is *well-founded*, if there does not exist an infinite sequence $a_0, a_1, \ldots$, where $a_i \in \mathcal{A}$, such that $a_i \succ a_{i+1}$ for all $i = 0, 1, \ldots$.

For the relation $\succ$, we denote by $\prec$ its inverse relation, i.e.,

$$a \prec b \iff b \succ a,$$

and by $\preceq$ the reflexive extension

$$a \preceq b \iff (a \prec b) \text{ or } (a = b).$$

Assume a well-founded relation $(\mathcal{A},\succ)$, and a partial *ranking function* $\delta : \Sigma \mapsto \mathcal{A}$, mapping states into the domain $\mathcal{A}$. We denote the fact that δ is defined by $\delta \in \mathcal{A}$. The following rule uses well-founded induction to establish an extended response property.

$$\boxed{\textbf{WELL-RESP}\quad \begin{array}{ll} \text{W1.} & p \Rightarrow (q \vee \varphi) \\ \text{W2.} & \varphi \Rightarrow (\delta \in \mathcal{A}) \\ \text{W3.} & [\varphi \wedge (\delta = \alpha)] \Rightarrow \Diamond[q \vee (\varphi \wedge (\delta \prec \alpha))] \\ \hline & p \Rightarrow \Diamond q \end{array}}$$

Premise W1 ensures that p entails that either q already holds, or φ is established. Premise W2 ensures that δ is defined as long as φ holds. Premise W3 guarantees that if φ holds with a certain rank α, then eventually we will reach a state, in which either q holds, or φ is maintained but with a rank lower than α. Since a well-founded ranking cannot go on decreasing forever, we must eventually reach a q-state.

The adequacy of this set of rules for proving response properties is established in Theorem 7.3 presented in Section 7, which states:

> *The rules given above are complete, relative to assertional validity, for proving the $\mathcal{P}$-validity of any response property.*

6 Rules for Progress

In this section we deal with *progress* properties, which are the properties that can be expressed by a formula of the form

$$\Box\Diamond p \vee \Diamond\Box q,$$

for some past formulae p and q. There are several alternative forms in which every progress property can be recast. They are given by

$$\Box\Diamond p \rightarrow \Box\Diamond q, \quad \text{or} \quad \Box\Diamond p \Rightarrow \Diamond q.$$

We prefer to work with an extended form of the last formula;

$$(p \wedge \Box\Diamond r) \Rightarrow \Diamond q.$$

This formula states that any occurrence of p, that is followed by infinitely many occurrences of r, must eventually be followed by an occurrence of q.

Progress under Continual Fairness

If we work only under the assumption of continual fairness, that is, the family of recurrent fairness requirements happens to be empty, then we can base the proof of progress properties on some response properties and a well-founded argument. This is given by the C-PROG rule.

$$\boxed{\begin{array}{lll} \textbf{C-PROG} & \text{C1.} & p \Rightarrow (q \vee \varphi) \\ & \text{C2.} & \varphi \Rightarrow (\delta \in \mathcal{A}) \\ & \text{C3.} & [\varphi \wedge (\delta = \alpha)] \Rightarrow [(\varphi \wedge (\delta \preceq \alpha)) \,\mathsf{U}\, q] \\ & \text{C4.} & [r \wedge \varphi \wedge (\delta = \alpha)] \Rightarrow \Diamond[q \vee (\delta \prec \alpha)] \\ \hline & & (p \wedge \Box\Diamond r) \Rightarrow \Diamond q \end{array}}$$

Note that this rule uses the *Unless* operator U.

Premise C1 of the rule ensures that any position that satisfies p, either already satisfies q, or satisfies φ. Premise C2 ensures that δ is defined as long as φ holds. Premise C3 ensures that, starting at a position satisfying φ and having a defined rank α, φ is continuously maintained and the rank never increases above α until q occurs, if ever. Premise C4 indicates that an additional occurrence of r strengthens the non-increase, guaranteed by C3, into a guaranteed eventual decrease. Thus, if there are infinitely many occurrences of r then, either δ decreases infinitely often, which is impossible due to well-foundedness, or q is eventually realized.

The adequacy of this rule is stated by Corollary 7.1, presented in Section 7, which claims:

> *For a program with no recurrent fairness requirements, the* C-PROG *rule is complete, relative to assertional validity, for proving the $\mathcal{P}$-validity of any progress property.*

Obviously, a progress property $(p \wedge \Box\Diamond r) \Rightarrow \Diamond q$ can be valid over a program due to the fact that the simpler response property $p \Rightarrow \Diamond q$ is valid. The theorem above depends on a particular mechanism to guarantee that infinitely many occurrences of r cause the eventual occurrence of q. This mechanism is based on a ranking function, measuring the distance away from the realization of q, such that each occurrence of an extra r causes an eventual decrease in the rank.

Progress under Recurrent Fairness

When we have recurrent fairness requirements, a well-founded decrease is not the only mechanism by which infinitely many occurrences of r can cause the computation to progress from p to q. Another possible mechanism is based upon a recurrent fairness requirement $(E, T) \in \mathcal{R}$, such that each transition in T leads from p to q, and each occurrence of r causes E to eventually become enabled (at least once). Consequently, the rule C-PROG is no longer adequate.

To cover the case of recurrent fairness, we present first a single-step rule for progress under recurrent fairness. The rule concerns a recurrent fairness requirement $(E, T) \in \mathcal{R}$, and past formulae p, r, q, and φ.

$$\boxed{\begin{array}{lll}
\textbf{R-PROG} & \text{R1.} & p \Rightarrow (q \vee \varphi) \\
 & \text{R2.} & \{\varphi\} T \{q \vee \varphi\} \\
 & \text{R3.} & \{\varphi\} T \{q\} \\
 & \text{R4.} & [\varphi \wedge \Box\Diamond(\varphi \wedge r)] \Rightarrow \Diamond(q \vee En(E)) \\
\hline
 & & (p \wedge \Box\Diamond r) \Rightarrow \Diamond q
\end{array}}$$

This rule establishes a single-step progress, under the assumption of the recurrent fairness requirement $(E, T) \in \mathcal{R}$. Several single-step progress prcperties can be combined, using the properties of monotonicity and transitivity of the progress formula. Below we present two rules, properly belonging to the general part, for these two properties.

$$\boxed{\begin{array}{ll}
\textbf{P-MON} & p' \Rightarrow p\ ,\ r' \Rightarrow r\ ,\ q \Rightarrow q' \\
 & (p \wedge \Box\Diamond r) \Rightarrow \Diamond q \\
\hline
 & (p' \wedge \Box\Diamond r') \Rightarrow \Diamond q'
\end{array}}
\qquad
\boxed{\begin{array}{ll}
\textbf{P-TRNS} & (p \wedge \Box\Diamond r) \Rightarrow \Diamond q \\
 & (q \wedge \Box\Diamond r) \Rightarrow \Diamond t \\
\hline
 & (p \wedge \Box\Diamond r) \Rightarrow \Diamond t
\end{array}}$$

Finally, we have a well-founded rule for combining together progress properties using induction.

$$\begin{array}{lll} \textsc{well-prog} & \text{W1.} & p \Rightarrow (q \vee \varphi) \\ & \text{W2.} & \varphi \Rightarrow (\delta \in \mathcal{A}) \\ & \text{W3.} & [\varphi \wedge (\delta = \alpha) \wedge \Box\Diamond r] \Rightarrow \Diamond[q \vee (\varphi \wedge (\delta \prec \alpha))] \\ \hline & & [p \wedge \Box\Diamond r] \Rightarrow \Diamond q \end{array}$$

This more general case is summarized by Theorem 7.4 presented in Section 7.

The rules given above are complete, relative to assertional validity, for proving the $\mathcal{P}$-validity of any progress property.

7 Completeness of the System

In this section we sketch the general ideas that lead to the (relative) completeness of the rules presented earlier. Since the most innovative part of the proof system presented in this paper is the incorporation of past formulae, we structure the completeness proof into two major steps, the first of which is the *elimination* of the past. The second step is left to deal with the restricted case of safety, response, and progress properties, where the subformulae p and q are only *state* formulae.

Encoding Past Formulae

We define a temporal formula as *stratified* if it contains no future operator within the scope of a past operator. Obviously, all formulae in canonical form are stratified, because they never apply past operators to strict-future formulae.

Let us fix our attention on a program P and a stratified formula φ, whose validity over P we wish to establish.

Define Φ to be the set of subformulae of φ (possibly including φ) whose principal operator is a past operator, i.e., $\odot$ or $\mathcal{S}$. We define a set of new boolean variables $\mathcal{B}$ consisting of a variable b_p for each formula $p \in \Phi$. We intend to use the variable b_p to encode p, i.e., as a variable that will be true at a position in a computation iff the formula p is true there.

Let q be a subformula of φ, and p a subformula of q. We define p to be Φ-*maximal* in q if

- $p \in \Phi$ and
- there is no r, another subformula of q, such that $r \in \Phi$ and p is a proper subformula of r, i.e., strictly contained in r.

Let $p_1, \ldots, p_n$ be all the Φ-maximal subformulae of q. We define the *statification* (i.e., encoding of past formulae as state formulae) of q, denoted by $stat(q)$ (or q_s), to be

$$stat(q) : q[b_{p_1}/p_1, \ldots, b_{p_n}/p_n].$$

That is, $stat(q)$ is obtained from q by replacing all occurrences of the subformula p_i by the variable b_{p_i}, for $i = 1, \ldots, n$. It is not difficult to see that, in the special case that q is a past formula, $stat(q)$ is a state formula.

Replacing past formulae by boolean variables is obviously not enough, unless we can guarantee that in all positions of the computation the variable b_p assumes the same truth value as p. To achieve this we modify the program P, given by the system $\langle V, \Sigma, \mathcal{T}, \Theta, \mathcal{C}, \mathcal{R}\rangle$, to obtain its *statified* version P_s, given by $\langle \hat{V}, \hat{\Sigma}, \hat{\mathcal{T}}, \hat{\Theta}, \hat{\mathcal{C}}, \hat{\mathcal{R}}\rangle$, where we define:

- $\hat{V} = V \cup \mathcal{B}$. That is, we augment V by the new boolean variables in $\mathcal{B}$.
- $\hat{\Sigma}$ – The set of interpretations over $\hat{V}$. Variables in $\mathcal{B}$ should be assigned boolean values.
- $\hat{\mathcal{T}}$ – Corresponding to each $\tau \in \mathcal{T}$, we place in $\hat{\mathcal{T}}$ a transition $\hat{\tau}$, whose transition relation is given by $\hat{\rho_\tau} = \rho_\tau \wedge N$. The assertion $N(\hat{V}, \hat{V}')$ controls the evolution of the variables in $\mathcal{B}$ between each state and its successor, and ensures that it corresponds to the evolution of the past formulae they stand for. The assertion N is a conjunction containing a conjunct $C(p)$ for each $p \in \Phi$. These conjuncts are given by:
  - $C(\ominus p)\ :\ b'_{\ominus p} \equiv stat(p)$.
    This conjunct guarantees that the boolean value of $b_{\ominus p}$ in the next state equals the truth-value of $stat(p)$ in the current state.
  - $C(p\mathcal{S}q)\ :\ b'_{p\mathcal{S}_q} \equiv [(stat(q))' \vee (b_{p\mathcal{S}_q} \wedge (stat(p))')]$.
    This conjunct guarantees that $b_{p\mathcal{S}_q}$ is true in the next state iff either $stat(q)$ holds there, or $stat(p)$ holds there and $b_{p\mathcal{S}_q}$ holds now.
- $\hat{\Theta}\ :\ \Theta \wedge Init$. The assertion $Init$ ensures that the initial value of each variable $b_p \in \mathcal{B}$ matches the initial value of the past formula p. The assertion $Init$ contains a conjunct $\mathcal{I}(p)$ for each $p \in \Phi$, given by:
  - $\mathcal{I}(\ominus p)\ :\ \neg b_{\ominus p}$.
    This conjunct states that all *previous* formulae are initially false.
  - $\mathcal{I}(p\mathcal{S}q)\ :\ b_{p\mathcal{S}_q} \equiv stat(q)$.
    This conjunct states that the only way for $p\mathcal{S}q$ to hold at the first state in a computation is for $stat(q)$ to hold there.

The structure of the fairness families $\hat{\mathcal{C}}$ and $\hat{\mathcal{R}}$ is identical to that of $\mathcal{C}$ and $\mathcal{R}$, except for the trivial renaming of each τ to $\hat{\tau}$.

Example 7.1 Consider the simple program, presented in Example 4 above, which was given by $V = \{x\}$, $\mathcal{T} = \{\tau\}$, where $\rho_\tau : x' = x + 1$, and $\Theta : x = 0$. The formula considered there is

$$\varphi\ :\ \Box((x = 10) \rightarrow \diamondminus(x = 5)).$$

Clearly, for this case $\Phi = \{\diamondminus(x = 5)\}$, yielding a single boolean variable b, corresponding to the past formula $\diamondminus(x = 5)$, which is an abbreviation for $\mathbf{T}\mathcal{S}(x = 5)$. Consequently, we have $stat(\varphi) : \Box((x = 10) \rightarrow b)$, and the statified program P_s is given by:

- $\hat{V} = \{x, b\}$.
- $\hat{\mathcal{T}} = \{\hat{\tau}\}$, where (following some simplifications) $\hat{\rho_\tau}\ :\ (x' = x + 1) \wedge (b' \equiv [(x' = 5) \vee b])$.

- $\hat{\Theta}$: $(x = 0) \wedge (b \equiv (x = 5))$, which is equivalent to $(x = 0) \wedge \neg b$.

Theorem 7.1 (Past Elimination)

- *The formula φ is valid over P iff $\varphi_s = stat(\varphi)$ is valid over P_s.*
- *Any proof of $P_s \vdash \varphi_s$, using the proof system presented in this paper, can be effectively transformed to a proof of $P \vdash \varphi$.*

Proof: The first statement of the theorem follows from the fact that there is a one-to-one correspondence between computations of P and computations of P_s, such that for every σ, a computation of P, and $\hat{\sigma}$, the corresponding computation of P_s, position j, and past formula $p \in \Phi$:

$$(\sigma, j) \models p \quad \Longleftrightarrow \quad (\hat{\sigma}, j) \models (b_p = \mathrm{T}).$$

This fact can be proved by induction on $j = 0, 1, ...$ and structural induction on $p \in \Phi$.

The second statement of the theorem is proven by showing that, replacing each line $\vdash \psi$ in the proof of $P_s \vdash \varphi_s$ by the line $\vdash stat^{-1}(\psi)$, we obtain a sound proof of $P \vdash \varphi$. The transformation $stat^{-1}(\psi)$ replaces each occurrence of b_p in ψ by the past formula p, each occurrence of b'_p by p', and each occurrence of $\hat{\Theta}$ and $\hat{\rho}_\tau$ by Θ and ρ_τ, respectively.

A detailed proof of this fact considers the different justifications for the line $\vdash \psi$, and shows the corresponding justifications for $\vdash stat^{-1}(\psi)$.

An illustrative case in point is a proof line stating the validity of the verification condition $\{\mathrm{T}\}\hat{\tau}\{b_{pSq}\}$, for the simple case that p and q are state formulae, and that the line is justified by generalization of a valid state formula.

This leads to the proof line

$$\vdash \; \hat{\rho}_\tau \Rightarrow b'_{pSq},$$

which can be written as

$$\vdash \; [\rho_\tau \;\wedge\; (b'_{pSq} \equiv [q' \vee (b_{pSq} \wedge p')])] \;\Rightarrow\; b'_{pSq},$$

which is equivalent to

$$\vdash \; \rho_\tau \;\Rightarrow\; [q' \vee (b_{pSq} \wedge p')].$$

Since ρ_τ does not refer to b_{pSq}, this line can be valid only if $\rho_\tau \rightarrow q'$ is a valid state formula. Applying $stat^{-1}$ to $\hat{\rho}_\tau \Rightarrow b'_{pSq}$, we obtain

$$\vdash \; \rho_\tau \Rightarrow (pSq)',$$

which expands to

$$\vdash \; \rho_\tau \;\Rightarrow\; [q' \vee ((pSq) \wedge p')].$$

Clearly, the validity of $\rho_\tau \rightarrow q'$, claimed above, can be used to justify this line.

A small technical problem is that a naive substitution of a past formula p for the variable b_p may result in formulae that are not allowed in our syntax. A case in point is a state formula $\alpha(b_p)$, in which the variable b_p falls in the scope of a quantification (on some other variable). Our syntax does not allow quantification over temporal formulae that are not state formulae. To resolve this problem, we observe that the state formula $\alpha(b_p)$ is equivalent, in all contexts, to the formula $(b_p \wedge \alpha(\mathrm{T})) \vee (\neg b_p \wedge \alpha(\mathrm{F}))$, in which the occurrences of b_p are outside any scopes of quantifications performed in α. Substitution in this latter form will result in a formula that is allowed by our syntax.

We should emphasize that the systematic elimination of the past from formulae and proofs, which facilitates establishing the completeness of the proof system, is not necessarily the approach we recommend for the actual verification of concrete programs. On the contrary; we strongly recommend working directly with past formulae which explicitly represent the relevant facts about the history of the computation leading to the current state. For example, we find the invariant $\Box((x = 10) \rightarrow \diamondsuit(x = 5))$ much more appealing and explicit than the encoded version $\Box((x = 10) \rightarrow b)$, accompanied by the tacit understanding that $b = \mathrm{T}$ iff we have passed in the past through a state in which $x = 5$.

Having shown how the past can be systematically eliminated, and replaced by state formulae, it only remains to show that the rules given above are adequate for proving the validity of the three classes of formulae:

$$\Box p \qquad p \Rightarrow \Diamond q \qquad (p \wedge \Box\Diamond r) \Rightarrow \Diamond q,$$

for the restricted case that p, q, and r are *state* formulae. These cases are more familiar, and the completeness of similar rules, for the cases of the safety and response classes, has been previously discussed in several places, such as [LPS81], [GFMdR85], [Fra86], [AS89], and [MP87].

Safety

Since we have restricted our attention to state formulae, it is sufficient to show that, whenever $\Box q$ is valid over the program P, we can prove this fact, using the INV rule. Premise I3 is proven by showing that $(\rho_\tau \wedge \varphi) \rightarrow \varphi'$ is a valid state formula for every $\tau \in \mathcal{T}$.

Theorem 7.2 (Completeness of Safety) *The rule* INV *is complete, relative to assertional validity, for proving the validity of safety formulae of the form $\Box q$, where q is a state formula.*

Proof: The basic idea of the proof is the construction of an assertion χ that holds in a state s iff s is *accessible*, i.e., appears in some computation of P. We then show *semantically* that, if $\Box q$ is indeed valid over P, then the premises of the INV rule are valid when taking χ for φ.

We assume that our data domain is expressive enough to encode *records* (i.e., lists) of data elements, and lists of records. In the definition of the assertion, we freely use the auxiliary variable r ranging over records, and a variable λ ranging over lists of records. We are mainly interested in records r of size $|V|$, and often write $r = V$ to denote that the record r contains a list of elements equal to the current values of the state variables V. We use the subscripted expression $\lambda[i]$ to refer to the i-th element of λ, and the expression $last(\lambda)$ to refer to the last element of λ. For an assertion $\varphi(V)$, referring to the state variables V, and a record r of size equal to that of V, we denote by $\varphi(r)$ the assertion φ in which the value $r[i]$ is substituted for the state variable $u_i \in V$, for $i = 1, ..., |V|$.

The assertion χ is given by:

$$\chi(V): \quad \exists\lambda : \Big((|\lambda| > 0) \wedge \alpha \wedge \beta \wedge \gamma\Big).$$

The body of the assertion χ (to which we refer as $\Psi(V, \lambda)$) consists, in addition to the requirement that λ is non-empty, of three clauses, given by:

$$\alpha: \quad \Theta(\lambda[1])$$

$$\beta: \quad V = last(\lambda)$$

$$\gamma: \quad \forall i (1 \leq i < |\lambda|): \bigvee_{\tau \in T} \rho_\tau(\lambda[i], \lambda[i+1]).$$

The assertion χ states the existence of a list of records λ of length $n = |\lambda| > 0$. The list λ encodes the history of a computation from some initial state to the current state. Each element $\lambda[i]$, $i = 1, \ldots, n$, is a record of data elements, representing the values of the state variables V at the i-th state of the computation.

Clause α states that $\lambda[1]$ satisfies Θ, the initial assertion of the program.

Clause β states that the current state variables V equal $last(\lambda) = \lambda[n]$, the last record in λ.

Clause γ states that the $(i+1)$-st record of λ, for each $i = 1, \ldots, n-1$, is a τ-successor of the i-th record, for some transition τ, guaranteeing the correct succession from $\lambda[1]$ to $\lambda[n]$.

We will show now that χ, when substituted for φ, validates the three premises of the **INV** rule.

I1. $\Theta \rightarrow \chi$
It is not difficult to see that taking λ to be (V), i.e., the list consisting of the single record containing the current values of $u_1, \ldots, u_{|V|}$, the assertion $\Theta(V)$ implies the body $\Psi(V, \lambda)$.

I2. $\chi \rightarrow q$
By our assumption that $\Box q$ is valid over P, it follows that each accessible state satisfies q. Since χ characterizes precisely the accessible states, the premise follows.

I3. $[\rho_\tau(V, V') \wedge \exists \lambda : \Psi(V, \lambda)] \rightarrow \exists \lambda' : \Psi(V', \lambda')$, for each $\tau \in T$.
It is not too difficult to see that if V, V', and λ satisfy $\rho_\tau(V, V') \wedge \Psi(V, \lambda)$, then there exists a λ' which satisfies $\Psi(V', \lambda')$. An appropriate choice is

$$\lambda': \quad \lambda * \langle V' \rangle,$$

i.e., the list obtained by appending to the end of λ an additional record, consisting of the list of the values of the primed variables V'.

Since we are interested in showing completeness, relative to assertional validity, it is sufficient to show that the premises are assertionally *valid*, as we have done above.

Response

As a complete rule for establishing response properties of the form $p \Rightarrow \Diamond q$, for the restricted case that p and q are state formula, we propose the following **F-RESP** rule, which is an appropriate combination of the **WELL-RESP**, **C-RESP**, and **R-RESP** rules. As usual, the rule stipulates the existence of an auxiliary assertion φ, a well-founded relation $(\mathcal{A}, \succ)$, and a partial ranking function $\delta : \Sigma \mapsto \mathcal{A}$, mapping states into the domain $\mathcal{A}$.

Since we intend to combine together continual and recurrent fairness, it is helpful to form the union of the continual and recurrent fairness requirements into one set of fairness requirements $\mathcal{F} = \mathcal{C} \cup \mathcal{R}$.

F-RESP F1. $p \Rightarrow (q \vee \varphi)$

F2. $\varphi \Rightarrow (\delta \in \mathcal{A})$

F3. $\{\varphi \wedge (\delta = \alpha)\}\ \mathcal{T}\ \{q \vee (\varphi \wedge (\delta \preceq \alpha))\}$

For each $\alpha \in \mathcal{A}$, there exists a fairness requirement $F_\alpha = (E_\alpha, T_\alpha) \in \mathcal{F}$, such that

F4. $\{\varphi \wedge (\delta = \alpha)\}\ T_\alpha\ \{q \vee (\varphi \wedge (\delta \prec \alpha))\}$

If $F_\alpha \in \mathcal{C}$, then

C5. $[\varphi \wedge (\delta = \alpha)] \Rightarrow (q \vee En(E_\alpha))$

If $F_\alpha \in \mathcal{R}$, then

R5. $\mathcal{F} - \{F_\alpha\} \vdash$

$$\frac{[\varphi \wedge (\delta = \alpha)] \Rightarrow \Diamond\big[q \vee (\varphi \wedge (\delta \prec \alpha)) \vee En(E_\alpha)\big]}{p \Rightarrow \Diamond q}$$

This rule combines well-foundedness with single-step rules. For each parameter $\alpha \in \mathcal{A}$, the rule requires the identification of a fairness requirement (E_α, T_α), that can be either a continual fairness or a recurrent fairness requirement. In both cases, it is required (by premise F4) that any transition in T_α leads from each φ-state s with rank α to a state s', that either satisfies q, or satisfies φ with a rank strictly lower than α. Any transition not in T_α is required (by premise F3) to lead from each φ-state with rank α to a state s', that either satisfies q, or satisfies φ with a rank not higher than α.

For the case that (E_α, T_α) is a continual fairness requirement, premise C5 requires that each φ-state with rank α, either satisfies q, or enables E_α. For the case that (E_α, T_α) is a recurrent fairness requirement, premise R5 requires that each φ-state s with rank α is eventually followed by a state s', that either satisfies q, or satisfies φ with a rank lower than α, or enables E_α. To avoid circularity, premise R5 is to be proven for a simpler program, in which $F_\alpha = (E_\alpha, T_\alpha)$ is removed from the list of fairness requirements. This is feasible because when trying to achieve a state in which E_α is enabled, we cannot be helped by any transition of E_α, since its activation from a state s' implies that E_α is already enabled on s'.

The following lemma establishes a connection between an arbitrary well-founded relation and a well-founded ranking. Such a ranking is required for the rule F-RESP.

Lemma 7.1 *Let B be a well-founded relation over the set S. Then there exists a* total *ranking function $\delta : S \mapsto Ordinals$, mapping each element of S into some ordinal, such that:*

a. $sBs' \rightarrow \delta(s) > \delta(s')$.

b. If $s'Bs'' \rightarrow sBs''$ for every $s'' \in S$, then $\delta(s) \geq \delta(s')$.

Based on this lemma, we can now state and prove the main completeness theorem.

Theorem 7.3 (Completeness for Response) *The rule* F-RESP *is complete, relative to assertional validity, for proving the validity of response formulae of the form $p \Rightarrow \Diamond q$, where p and q are state formulae.*

Proof: Assume the formula $p \Rightarrow \Diamond q$ to be valid over the program P. We have to show the existence of an appropriate assertion φ, a well-founded ordering $(\mathcal{A}, \succ)$, a ranking function $\delta : \Sigma \mapsto \mathcal{A}$, and a selection function, identifying for each $\alpha \in \mathcal{A}$ a fairness requirement $F_\alpha = (E_\alpha, T_\alpha) \in \mathcal{F}$, such that together they satisfy the premises of the F-RESP rule. Due to the incrementality principle, it is sufficient to show for each premise ψ, the validity of $\chi \rightarrow \psi$, where χ is the assertion characterizing accessibility, and whose invariance over P has been established by the preceding theorem.

We define a (computation) *segment* to be a finite sequence of states $\sigma : s_1, s_2, ..., s_k$, for $k \geq 1$, such that for every $i = 1, ..., k-1$, s_{i+1} is a τ-successor of s_i, for some $\tau \in \mathcal{T}$. We say that the segment σ *departs* from s_1, and that it *connects* s_1 to s_k. We define a segment to be *q-free* if none of the states $s_1, ..., s_k$ satisfies q. From now on, when we refer to a segment, we mean a q-free segment.

We define the assertion φ required by the F-RESP rule as follows.

$s \models \varphi \iff$ There exists an accessible p-state $\hat{s}$ and a q-free segment, connecting $\hat{s}$ to s.

This definition is verbal, but it is clear how it can be expressed in our assertion language, using techniques similar to the ones used for defining χ in the theorem about safety.

It is clear that if the state s satisfies φ, and some computation contains s at position j, then, due to the assumed validity of $p \Rightarrow \Diamond q$ there must be a later position $k \geq j$ satisfying q.

It is also obvious that φ, defined in this way, satisfies premise F1 of the rule, i.e., $p \Rightarrow (q \vee \varphi)$. This is because, if s is an accessible p-state which does not satisfy q, then we can take $\hat{s} = s$ and the singleton segment s, connecting s to itself, as a justification for the claim that s satisfies φ. We can restrict our considerations here and elsewhere to accessible states only due to the incrementality principle.

Let the family of combined fairness requirements $\mathcal{F}$ consists of the sets $F_1, ..., F_m$, where each F_i is either a continual fairness requirement or a recurrent fairness requirement. Without loss of generality, assume that $F_1 = (\mathcal{T}, \mathcal{T})$ is a continual fairness requirement, consisting of a pair of sets, each being the full set of transitions $\mathcal{T}$. For a segment $\sigma : s_1, ..., s_k$ and a fairness requirement $F_i \in \mathcal{F}$, we say that $F_i = (E_i, T_i)$ is *fulfilled* in σ if one of the following holds

- Some transition of T_i is taken in σ.
- F_i is a continual fairness requirement, and E_i is disabled on some state in σ.

For a segment σ, we define $sat(\sigma)$ to be the set of all indices $i = 1, ..., m$ such that F_i is fulfilled in σ. Let Φ denote the set of all states satisfying φ. We define a binary relation B on Φ by:

$s B \bar{s} \iff$ There exists a q-free segment σ connecting s to $\bar{s}$, such that $sat(\sigma) = \{1, ..., m\}$.

We claim that B is a well-founded relation over Φ. This is because an infinite sequence

$$s^1 \; B \; s^2 \; B \; s^3 ...,$$

gives rise to a computation

$$\sigma \; : \; s^0, ..., \hat{s}, ..., s^1, ..., s^2, ..., s^3, ...,$$

such that s^0 is initial, $\hat{s}$ satisfies p, and no state beyond $\hat{s}$ satisfies q. Such a computation obviously violates our assumption that $p \Rightarrow \Diamond q$ is valid over P. The fact that the sequence above is a computation, in particular that it satisfies all the fairness requirements, hinges on the assumption that the satisfiability set of each segment $s^i, ..., s^{i+1}$ is the full set $\{1, ..., m\}$.

According to Lemma 7.1, there exists a ranking function $\delta_0 : \Phi \mapsto \mathit{Ordinals}$, mapping states in Φ into the ordinals.

Let s be a φ-state and s' a successor of s. If s' does not satisfy q, then it is also a φ-state. In this case we show that $\delta_0(s) \geq \delta_0(s')$. This inequality is ensured by clause b of Lemma 7.1, provided we show that for every s'', $s'Bs''$ implies sBs''.

Indeed, let s'' be a state such that $s'Bs''$. By the definition there exists a segment $\sigma' : s', ..., s''$ connecting s' to s'', such that $sat(\sigma') = \{1, ..., m\}$. It is obvious that the segment $\sigma : s, s', ..., s''$, formed by appending s to the beginning of s', connects s to s'', and that $sat(\sigma) = \{1, ..., m\}$. This establishes sBs''.

The ranking δ_0 is not fine enough to uniquely identify the fairness set F_a. We therefore augment it by a secondary ranking δ_1 defined as follows.

For a segment σ, we define the *deficit* of σ, denoted by $\Delta(\sigma)$, to be the smallest positive integer i, such that F_i is not fulfilled in σ. In the case that $sat(\sigma) = \{1, ..., m\}$, $\Delta(\sigma)$ is defined to be $m + 1$. We define a segment $\sigma : s_1, ..., s_k$ to be *leveled* if $\delta_0(s_1) = ... = \delta_0(s_k)$.

For every φ-state s, we define its secondary ranking $\delta_1(s)$ by

$$\delta_1(s) \ = \ max\{\Delta(\sigma) \mid \sigma \text{ is a leveled segment departing from } s \ \}.$$

The complete ranking function, to be used in the rule, is formed by the lexicographical pairing $\delta(s) = (\delta_0(s), \delta_1(s))$. The range of the function δ is defined to be $\mathcal{A}$, the set of all pairs of the form (α_0, i), where α_0 is an ordinal and $i \leq m + 1$.

The ordering $\succ$ over $\mathcal{A}$ is defined by

$$(\alpha_0, i) \succ (\alpha_0', i') \quad \Longleftrightarrow \quad (\alpha_0 > \alpha_0') \vee \Big((\alpha_0 = \alpha_0') \wedge (i > i')\Big)$$

. Clearly, this ordering is well-founded.

There are several properties these ranking functions satisfy.

P1. For every φ-state s, $\delta_1(s) \leq m$.
Let σ be a leveled segment connecting s to some s'. If $sat(\sigma)$ equals $\{1, ..., m\}$, then sBs' holds, which leads to $\delta_0(s) > \delta_0(s')$, contradicting the fact that σ is leveled. It follows that at least some F_i is not fulfilled in σ, and therefore $\delta_1(s) \leq m$.

P2. For every φ-state s and its successor s', either s' satisfies q, or $\delta(s) \succeq \delta(s')$.
Assume that s' does not satisfy q. We have already shown that $\delta_0(s) \geq \delta_0(s')$. If $\delta_0(s) > \delta_0(s')$, then clearly $\delta(s) \succeq \delta(s')$. In the other case, i.e., $\delta_0(s) = \delta_0(s')$, let $\delta_1(s')$ be $i \leq m$. By the definition of δ_1, there exists a leveled segment $\sigma' : s', ..., s''$, such that i is the smallest index of a fairness requirement F_i, which is not fulfilled in σ'. Consider the augmented segment $\sigma : s, s', ..., s''$. Clearly, σ is leveled and any F_i fulfilled in σ' is also fulfilled in σ'. It follows that the deficit of σ, $\Delta(\sigma) \geq \Delta(\sigma') = i$. Since σ is only one of the leveled segments departing from s, and $\delta_1(s)$ is defined to be the maximum of the deficits of all such segments, it follows that $\delta_1(s) \geq i$.

P3. Let s be a φ-state, such that $\delta_1(s) = i$. Let s' be a τ-successor of s, where τ is one of the transitions of T_i. Then, either s' satisfies q, or $\delta(s) \succ \delta(s')$.

It is sufficient to consider the case that s' does not satisfy q and that $\delta_0(s) = \delta_0(s')$, and to show that $\delta_1(s) > \delta_1(s')$. Assume, to the contrary, that $\delta_1(s) = \delta_1(s') = i$. Let $\sigma' : s', ..., s''$ be, as before, the segment realizing the deficit i for s'. Clearly, the augmented segment $\sigma : s, s', ..., s''$ fulfills all the requirements fulfilled by σ', and in addition also fulfills F_i. It follows that $\Delta(\sigma) > i$, and therefore also $\delta_1(s) > i$, contradicting our original assumptions.

We proceed to show that all the premises of the **F-RESP** are satisfied by these definitions. We have already shown that F1 is valid.

F2. $\varphi \Rightarrow (\delta \in \mathcal{A})$

Clearly δ_0 and δ_1 are defined on every φ-state. It follows that δ is also defined.

For the next premises, we identify for each value $\alpha = (\alpha_0, i) \in \mathcal{A}$, the *helpful fairness requirement* $F_\alpha = (E_\alpha, T_\alpha)$ to be $F_i = (E_i, T_i)$.

F3. $\{\varphi \wedge (\delta = \alpha)\}\ \mathcal{T}\ \{q \vee \big(\varphi \wedge (\delta \preceq \alpha)\big)\}$

It is straightforward to show that if s' is a successor of a φ-state s, then either s' satisfies q or it is also a φ-state, which by property P2 above satisfies $\delta(s) \succeq \delta(s')$.

F4. $\{\varphi \wedge (\delta = \alpha)\}\ T_\alpha\ \{q \vee \big(\varphi \wedge (\delta \prec \alpha)\big)\}$

Let s be a φ-state, such that $\delta_1(s) = i$, and s' a τ-successor of s, for some transition $\tau \in T_i$. If s' does not satisfy q, then it clearly satisfies φ, and by the property P3 stated above, also satisfies $\delta(s) \succ \delta(s')$.

For the case that $F_i = (E_i, T_i)$ is a continual fairness requirement, we proceed to show

C5. $[\varphi \wedge (\delta = \alpha)] \Rightarrow (q \vee En(E_\alpha))$

Let s be a φ-state, not satisfying q, such that $\delta_1(s) = i$. Let $\sigma : s, ..., s''$ be the segment realizing the deficit i. If E_i were disabled on s, then according to the definition F_i would have been fulfilled in σ. We conclude that E_i must be enabled on s.

For the case that $F_i = (E_i, T_i)$ is a recurrent fairness requirement, we proceed to show

R5. $\mathcal{F} - \{F_\alpha\} \vdash [\varphi \wedge (\delta = \alpha)] \Rightarrow \Diamond[q \vee \big(\varphi \wedge (\delta \prec \alpha)\big) \vee En(E_\alpha)]$

Let P' denote the program which is identical to P in all components, except that the recurrent fairness requirement $F_i = F_\alpha$ has been removed from its combined fairness set $\mathcal{F}$. We proceed to show that $P' \models \psi$, where ψ is the state formula whose validity is claimed to be provable in R5. Assume to the contrary, that ψ is not valid over P'. In that case there must exists σ, a computation of P', containing at some position j a φ-state s with rank α (and $\delta_1(s) = i$), such that no position beyond j satisfies $q \vee \big(\varphi \wedge (\delta \prec \alpha)\big) \vee En(E_i)$. Being a computation of P' means that it satisfies all the fairness requirements posed by P, except possibly F_i. However, since $En(E_\alpha) = En(E_i)$ is one of the disjuncts excluded beyond position j, it follows that E_i is enabled only finitely many times on σ, which implies that σ is fair also with respect to F_i, and is therefore also a computation of P. This violates our original assumption that $p \Rightarrow \Diamond q$ is valid over P.

If we base our completeness proof on induction on the size of $\mathcal{F}$, the combined fairness set, we have just reduced the completeness problem of response properties for programs with $|\mathcal{F}| = n + 1$, to that of program with $|\mathcal{F}| = n$. By such an induction, since we have just shown that $P' \models \psi$, it follows that $P' \vdash \psi$, as is required by R5.

Note that the reduction implied by premise R5 always removes from $\mathcal{F}$ a *recurrent fairness* requirement. This implies that after any number of such removals $\mathcal{F}$ will still contain the continual fairness requirement $(\mathcal{T}, \mathcal{T})$, and therefore $|\mathcal{F}| \geq 1$.

It follows that the base case for the induction can be $|\mathcal{F}| = n = 1$. In this case, the only helpful requirement can be $(\mathcal{T}, \mathcal{T})$. The arguments above are fully applicable for this case, except that the case leading to R5 never arises, since the helpful requirement is always a continual requirement. ∎

Progress

Lastly, we consider proving the completeness of our proof system for proving formulae of the form $(p \wedge \Box\Diamond r) \Rightarrow \Diamond q$, for state formulae p, q, and r. A helpful intuition, which will guide us in the proof, is that such a formula is valid over P iff the response formula $p \Rightarrow \Diamond q$ is valid over a program P^{+} which differs from P by having an additional continual fairness requirement, which demands that every computation contains infinitely many r-states.

With this understanding, we proceed in a route very similar to that of establishing completeness for response properties. We consider first the general case of a program that has both continual and recurrent fairness requirements.

As a first step, we formulate a combined rule for progress, using a notation similar to that of the F-RESP rule, with some small changes. We define the combined fairness set $\mathcal{F}_r = \{(\phi, \mathcal{T}_P)\} \cup \mathcal{C} \cup \mathcal{R}$. Thus, the set $\mathcal{F}_r$ contains, in addition to the continual fairness requirements taken from $\mathcal{C}$, and the recurrent fairness requirements taken from $\mathcal{R}$, also the special "fairness" requirement $(\phi, \mathcal{T}_P)$. This virtual fairness requirement contains no transitions in its E set, but restricts our attention (as may be seen from the rule) to computations, in which r occurs infinitely many times. We represent the requirements contained in $\mathcal{F}_r$ by the list $F_0, F_1, ..., F_m$, where $F_1, ..., F_m$ are the real fairness requirements, and $F_0 = (\phi, \mathcal{T}_P)$ is the virtual one. Following is the combined rule for progress.

F-PROG

F1. $p \Rightarrow (q \vee \varphi)$

F2. $\varphi \Rightarrow (\delta \in \mathcal{A})$

F3. $\{\varphi \wedge (\delta = \alpha)\}\ \mathcal{T}\ \{q \vee \big(\varphi \wedge (\delta \preceq \alpha)\big)\}$

For each $\alpha \in \mathcal{A}$, there exists a fairness requirement $F_\alpha = (E_\alpha, T_\alpha) \in \mathcal{F}_r$, such that:

If $F_\alpha \neq (\phi, \mathcal{T}_P)$, then

F4. $\{\varphi \wedge (\delta = \alpha)\}\ T_\alpha\ \{q \vee \big(\varphi \wedge (\delta \prec \alpha)\big)\}$

If $F_\alpha = (\phi, \mathcal{T}_P)$, then

F5. $\{\varphi \wedge (\delta = \alpha) \wedge r\}\ \mathcal{T}\ \{q \vee \big(\varphi \wedge (\delta \prec \alpha)\big)\}$

If $F_\alpha \in \mathcal{C}$, then

C6. $[\varphi \wedge (\delta = \alpha)] \Rightarrow (q \vee En(E_\alpha))$

If $F_\alpha \in \mathcal{R}$, then

R6. $\mathcal{F}_r - \{F_\alpha\} \vdash$

$[\varphi \wedge (\delta = \alpha) \wedge \Box\Diamond r] \Rightarrow \Diamond\Big[q \vee \big(\varphi \wedge (\delta \prec \alpha)\big) \vee En(E_\alpha)\Big]$

$(p \wedge \Box\Diamond r) \Rightarrow \Diamond q$

Theorem 7.4 (Completeness for Progress) *The rule* F-PROG *is complete, relative to assertional validity, for proving the validity of progress formulae of the form* $(p \wedge \Box\Diamond r) \Rightarrow \Diamond q$, *where* p, r, *and* q *are state formulae.*

Proof: Assume the formula $(p \wedge \Box\Diamond r) \Rightarrow \Diamond q$ to be valid over the program P. We adopt the definitions of φ, and q-free segments, from Theorem 7.3. We slightly modify the definition of fulfillment in a segment to read as follows:

For a segment $\sigma : s_1, ..., s_k$ and a fairness requirement $F_i = (E_i, T_i) \in \mathcal{F}_r$, we say that F_i is *fulfilled* in σ if one of the following holds:

- $i > 0$ and some transition of T_i is taken in σ.
- $i > 0$, F_i is a continual fairness requirement, and E_i is disabled on some state in σ.
- $i = 0$ and some state in σ satisfies r.

Thus, we associate the fulfillment of the set $F_0 = (\phi, \mathcal{T}_P)$ with the satisfaction of r. We define the set $sat(\sigma)$, for a segment σ, as before, except that its range may now be any subset of $\{0, 1, ..., m\}$. Similarly, we define the relation B to hold between two states, s and s', if there exists a segment σ, connecting them, such that $sat(\sigma) = \{0, 1, ..., m\}$. The relation B is well-founded, because an infinite sequence of B-related φ-states gives rise to a computation violating $(p \wedge \Box\Diamond r) \Rightarrow \Diamond q$. Consequently, we obtain the primary ranking δ_0. The definition of the *deficit* $\Delta(\sigma)$ of a segment σ is precisely the same as the corresponding definition in Theorem 7.3, except that it now ranges over $\{0, 1, ..., m\}$. This leads to the secondary ranking δ_1, and to the definition of the combined ranking $\delta = (\delta_0, \delta_1)$, which ranges over pairs (α_0, i), with α_0 an ordinal, and $0 \leq i \leq m$.

It is straightforward to verify that properties P1 and P2 are still valid, as is P3 for $\delta_1(s) = i > 0$. A special consequence of the definitions above is that if s is a φ-state, which satisfies r, then $\delta_1(s) > 0$.

We may now turn to establish the validity of the premises of the rule. Premises F1, F2, and F3, follow from arguments similar to the ones presented in the case of the response rule.

Given a parameter $\alpha = (\alpha_0, i)$, we identify the helpful fairness requirement F_α as $F_i \in \mathcal{F}_r$. Premise F4, which is applicable only in the case that $i > 0$, is justified by arguments similar to those of the response case. So are premises C6 and R6, which are also applicable only to the cases $i > 0$. Considering R6, the inductive argument has to consider a similar progress property for a simpler program.

Premise F5 holds trivially, since by the observation above, there can be no φ-state s, satisfying r, such that $i = \delta_1(s) = 0$. ∎

Using the constructions employed in the proof of this theorem, it is possible to derive the following corollary.

Corollary 7.1 (Completeness of Progress under Continual Fairness) *For a program with no recurrent fairness requirements, the* C-PROG *rule is complete, relative to assertional validity, for proving the $\mathcal{P}$-validity of any progress property.*

Proof: Assume the formula $p \Rightarrow \Diamond q$ to be valid over the program P, which has only continual fairness requirements. We adopt the definitions of the assertion φ, the ordering B, shown to be well-founded, and the ranking function δ_0, based on B, from the previous theorem. We take δ_0 for the ranking δ required by the C-PROG rule. It is not difficult to see that this choice of φ and δ satisfies premises C1–C3 of the rule. Let us consider premise C4. Assume a computation, in which the state s at position j satisfies $r \wedge \varphi$, and has the rank $\delta_0(s) = \alpha$. It is not difficult to see that there must be another state $\tilde{s}$, at position $k \geq j$, such that either $\tilde{s}$ satisfies q, or the segment $s \ldots \tilde{s}$ is q-free and fulfills all the (continual) fairness requirements associated with P. In the later case $sB\tilde{s}$ (since s satisfies φ), and according to clause a of Lemma 7.1, this implies that $\delta_0(s) > \delta_0(\tilde{s})$. This establishes premise C4.

Acknowledgement

We gratefully acknowledge the help of Alur Rajeev and Tom Henzinger in careful reading of the manuscript and thank them for many helpful suggestions.

References

[Apt81] K.R. Apt. Ten years of Hoare's logic: A survey - part I. *ACM Trans. Prog. Lang. Sys.*, 3:431–483, 1981.

[AS89] B. Alpern and F.B. Schneider. Verifying temporal properties without temporal logic. *ACM Trans. Prog. Lang. Sys.*, 11:147–167, 1989.

[Coo78] S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comp.*, 7:70–90, 1978.

[Fra86] N. Francez. *Fairness.* Springer, 1986.

[GFMdR85] O. Grumberg, N. Francez, J.A. Makowski, and W.-P. de Roever. A proof rule for fair termination. *Inf. and Comp.*, 66:83–101, 1985.

[Har79] D. Harel. *First-Order Dynamic Logic.* Lec. Notes in Comp. Sci. 68, Springer, 1979.

[Krö87] F. Kröger. *Temporal Logic of Programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science.* Springer, 1987.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *Proc. 8th Int. Colloq. Aut. Lang. Prog.*, pages 264-277. Lec. Notes in Comp. Sci. 115, Springer, 1981.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. Conf. Logics of Programs*, pages 196–218. Lec. Notes in Comp. Sci. 193, Springer, 1985.

[MP83a] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *Proc. 10th ACM Symp. Princ. of Prog. Lang.*, pages 141–154, 1983.

[MP83b] Z. Manna and A. Pnueli. Verification of concurrent programs: A temporal proof system. In J.W. DeBakker and J. Van Leuwen, editors, *Foundations of Computer Science IV, Distributed Systems: Part 2*, pages 163–255. Mathematical Centre Tracts 159, Center for Mathematics and Computer Science (CWI), Amsterdam, 1983.

[MP84] Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Sci. Comp. Prog.*, 32:257–289, 1984.

[MP87] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *Proc. 14th ACM Symp. Princ. of Prog. Lang.*, pages 1–12, 1987.

[MP89] Z. Manna and A. Pnueli. The anchored version of the tempoal framework. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 201–284. Lec. Notes in Comp. Sci. 354, Springer, 1989.

[OL82] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Prog. Lang. Sys.*, 4:455–495, 1982.

[Pnu86] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency*, pages 510–584. Lec. Notes in Comp. Sci. 224, Springer, 1986.

[Tho81] W. Thomas. A combinatorial approach to the theory of ω-automata. *Inf. and Cont.*, 48:261–283, 1981.